Supplementary Materials for:

A Grammar for Reproducible and Painless Extract-Transform-Load Operations on Medium Data

A Extended discussion of related work

In this section we summarize the major considerations that make the `etl` package a progressive step towards reproducible research on medium data for R users.

A.1 Reproducible research

To understand the current challenges we face in conducting reproducible research on PAMDAS, one must start with the notion of literate programming (Knuth 1984). In literate programming, source code is woven into an annotated narrative, so that one could read the source code and understand not just the code itself, but also how each piece of code fits into the larger design.

This idea leads to the notion of *reproducibility* in computational science. Donoho (2010) paraphrases Claerbout (1994):

> An article about a computational result is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result.

Ioannidis (2005) argues that most published research is false, and while his arguments are *statistical* rather than *computational*, they only help to underscore the importance of computational reproducibility.

In academia, a diverse set of fields including computer science (Donoho et al. 2009), economics (Ball & Medeiros 2012), archeology (Marwick 2017) and neuroscience (Eglen et al. 2017) are actively debating how they will recognize reproducible research. Organizations like Project TIER (`http://www.projecttier.org/`) and the Open Science Framework (`https://osf.io/`) provide protocols for conducting reproducible research, while statistics and data science educators are instilling reproducible practices in their students (Baumer et al. 2014). Top-tier journals like the *Journal of the American Statistical Association* have appointed reproducibility editors (Fuentes 2016).

Thus, while the need for research in all fields to be reproducible is clear, the specifications for what qualifies as reproducible are less clear, and the path towards achieving reproducibility is murkier still.

A.2 Medium data

In the past few years, *big data* has become an omnipresent buzzword that taps into our collective fascination with things that are massive. However, while a few enormous companies (e.g., Google, Facebook, Amazon, Walmart, etc.) generate and analyze truly big data (on the order of *exabytes* (EB), which are equal to 1000 *petabytes* (PB), which are equal to 1000 *terabytes* (TB), which are equal to 1000 *gigabytes* (GB)), most people who analyze data will never interact meaningfully with data of that size.

Most people will only encounter data that is *small* (a few gigabytes at most). These data fit effortlessly into a computer's memory, and thus the user experiences no challenges related to the data's size. Because a computer can access data in memory at lightning-fast speeds, efficient data analysis algorithms like searching ($O(n)$), sorting ($O(n \log n)$), and multiplying matrices (e.g., fitting a regression model) ($O(n^{2.376})$ (Williams 2012)) will run nearly instantly—even on a laptop. [1] Thus, for people working with small data, fundamental computer science concepts like the distinction between hardware and software, algorithmic efficiency, and bus speeds are immaterial.

For the vast majority of us who are unlikely to ever interact meaningfully with truly big data, *medium data* is both a viable solution and an accessible introduction to the challenges of big data (Horton et al. 2015). In Table 1, we constrast the relative sizes of data from the point of view of a personal computer user. Medium data is on the order of several gigabytes to a few terabytes. These data are large enough that they will not comfortably fit in memory on a personal computer without consequences, making a memory-only application like (vanilla) R a dubious candidate for data analysis. However, medium data are not so large they won't fit on a single hard disk, making them accessible to a single user without access to a computing cluster. An SQL-based RDBMS remains an appropriate storage and retrieval solution for medium data.

[1] Computer scientists use Big-O notation to describe the running time of algorithms by comparing the order of magnitude of the number of steps the algorithm takes to execute on an input of size n. An algorithm that runs in $O(n)$ time is *linear*, in the sense that the amount of time it will take to run is linearly proportional to the size of the input.

A.3 Existing challenges

The fundamental challenge of big data is scalability, but medium data comes with its own challenges. In the end, investment in properly setting up an RDBMS pays off in more efficient analysis.

First, everything with medium data takes a little longer, since the aforementioned algorithms are no longer instantaneous. A single line of code might take one minute to execute instead of a millisecond, but these brief delays compound. Thus, those who employ efficient code and workflows are rewarded for their efforts with shorter execution times.

Second, a data analyst has to know something about SQL administration in order to set up a database. Many introductory data science courses that teach SQL focus on writing `SELECT` queries to retrieve data from an existing database—not on writing table schemas and defining keys and indexes (Hardin et al. 2015).

Third, getting PAMDAS set up involves often laborious ETL operations. Downloading medium data is not instantaneous and is dependent on the speed of one's Internet connection. Wrangling data is notoriously time-consuming work: reasonable estimates suggest this may occupy as much as 50–80% of a data scientist's time.

For these reasons, a responsible data scientist will record their ETL operations in a script. But these scripts are often problematic, ad hoc solutions. Some common problems include:

Portability Shell scripts may not port across operating systems. While Apple's OS X operating system is POSIX-compliant, not all flavors of GNU/Linux are. Microsoft Windows requires additional software to implement a compatibility layer, and thus any such scripts are not likely to run on Windows without careful modification.

Usability Under time pressure, data scientists are likely to write scripts that work for them, and not necessarily for other people. Their scripts may be idiosyncratic and difficult for another person to use or modify.

Version Control Even if a data scientist uses a formal version control system like `git` and GitHub, a script that ran when it was written may not run at all points in the future.

Languages ETL scripts may be written in bash, Python, R, SQL, Perl, PHP, Ruby, Scala, Julia, or any combination of these languages and others. There may be good reasons for mixing different languages but ease of portability decreases with each additional language.

One recommended solution for bundling ETL scripts for R users is to create an R package (Wickham 2015). Packages provide users with software that extends the core functionality of R, and often data that illustrates the use of that functionality. R packages hosted on CRAN—the authoritative central repository—are checked for quality and documentation, helping to ensure their *usability*. Since R is cross-platform, these packages are *portable*. CRAN itself maintains distinct *versioning*, and while R packages are mostly written in R, there are a number of ways in which code from other *languages* can be embedded into an R package (e.g., Rcpp provides functionality to bundle C++ code (Eddelbuettel & François 2011)).

However, by design the types of data that can be contained in an R package hosted on CRAN are limited. First, packages are designed to be small, so that the amount of data stored in a package is supposed to be less than 5 *megabytes*. Furthermore, these data are static, in that CRAN allows only monthly releases. Alternative package repositories—such as GitHub—are also limited in their ability to store and deliver data that could be changing in real-time to R users. In Table 1 we contrast two different CRAN packages for on-time airline flight data (Wickham 2016, 2013), with an etl-dependent package that allows the user to build their own database of flight data (Baumer 2017). We note the change in scope that the airlines package allows: whereas the two existing data sets are restricted to small, static data from flights departing two Houston-area airports in 2011, or three New York City-area airports in 2013, respectively, the airlines package covers all domestic flights since 1987 departing from more than 350 airports nationwide, with more data available monthly.

Many R packages facilitate the retrieval of data from specific sources. In particular, the rOpenSci group maintains dozens of such packages (Boettiger et al. 2015). Other popular small CRAN packages that serve as APIs to large data sets include tigris (Walker & Rudis 2017) and UScensus2010 (Almquist 2010). While these packages are undoubtedly useful,

package	timespan	airports	size
hflights	2011	IAH, HOU	2.1 MB
nycflights13	2013	LGA, JFK, EWR	4.4 MB
airlines	1987–present	≈ 350	> 6 GB

Table 1: Alternative packaging of on-time flight data from the Bureau of Transportation Statistics in R. We note that the full scope of flight data is only accessible through the airlines package.

they are written by many different authors, and the syntax employed across packages varies greatly. In short, there is no consistent "grammar" (see Section 3). These packages are peripherals without a core.

Some dependency approaches do exist. Peng & Dominici (2008) illustrate how a small package for CRAN that interacts with large data repositories not hosted on CRAN could facilitate research in environmental epidemiology. These repositories are maintained by the package author through the use of a second package (Eckel & Peng 2009). More recently, the drat package provides a core that facilitates the creation of peripheral packages (Anderson & Eddelbuettel 2017). In this scheme the peripheral packages contain large amounts of data. The major drawback to both of these approaches is the requirement that the researcher maintain the large data repositories.

Boettiger (2015) advocates for the container-based solution Docker as an alternative packaging structure for reproducible research, and more recently Rocker (Boettiger & Eddelbuettel 2017), which provides Docker containers for R and RStudio. Çetinkaya-Rundel & Rundel (2017) promote this approach as university instructors. We see etl as fitting nicely into this paradigm, serving to further reduce barriers to reproducibility.

Perhaps the closest competitor to our approach is pitchRx (Sievert 2014), which performs ETL operations for a specific data set—in this case, detailed pitch information from Major League Baseball. Our approach places similar core functionality in the etl package and separates the data-source-specific functionality into small, easy-to-write packages that can be hosted on CRAN. The developer need not maintain any large data repositories—they need only to maintain the small bits of code that interact with the data provider. If, for any reason, the source data changes, etl users still retain copies of the raw data as they downloaded it.

We imagine that many of these aforementioned packages could be re-factored to have `etl` as a depedendency.

B A toy example

Here, we illustrate the functionality of the `etl` package on the built-in `mtcars` data set.

The first step is to instantiate an `etl` object using the `etl()` function. We use the `etl_create()` function to perform the entire ETL cycle on an object named `my_cars`. During this process, a local SQLite database is created in a temporary directory, that database is initialized, the `mtcars` data is "downloaded" (i.e., in this case, from memory), transformed, and finally uploaded to that same SQLite database.

```
library(etl)

## Loading required package:  dplyr
##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:stats':
##
##    filter, lag
## The following objects are masked from 'package:base':
##
##    intersect, setdiff, setequal, union

my_cars <- etl("mtcars") %>%
  etl_create()

## No database was specified so I created one for you at:
## /tmp/Rtmpfx3av7/file5b4c78fe7b0d.sqlite3
## Initializing DB using SQL script init.sqlite
## Extracting raw data...
## Transforming raw data...
## Loading 1 file(s) into the database...
```

The object my_cars is both an etl_mtcars object and a src_dbi object—and can thus do anything that any other src_dbi object can do. It also maintains a connection to the SQLite database, has two folders (e.g., raw and load) where it can store files, and knows about a table called mtcars that exists in the SQLite database.

```
class(my_cars)

## [1] "etl_mtcars" "etl"        "src_dbi"    "src_sql"    "src"

summary(my_cars)

## files:
##   n size                 path
## 1 1 0 GB  /tmp/Rtmpfx3av7/raw
## 2 1 0 GB /tmp/Rtmpfx3av7/load
##       Length Class            Mode
## con   1      SQLiteConnection S4
## disco 2      -none-           environment

my_cars

## dir:  2 files occupying 0 GB
## src:  sqlite 3.22.0 [/tmp/Rtmpfx3av7/file5b4c78fe7b0d.sqlite3]
## tbls: mtcars
```

Since my_cars is a DBI data source, the data stored in the SQLite database can be accessed in the usual manner. Here, we compute the average fuel economy for these cars. Note that these computations are performed by SQLite.

```
my_cars %>%
  tbl("mtcars") %>%
  group_by(cyl) %>%
  summarize(N = n(), mean_mpg = mean(mpg))
```

```
## Warning:  Missing values are always removed in SQL.
## Use `AVG(x, na.rm = TRUE)` to silence this warning

## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.22.0 [/tmp/Rtmpfx3av7/file5b4c78fe7b0d.sqlite3]
##     cyl     N mean_mpg
##   <int> <int>    <dbl>
## 1     4    11     26.7
## 2     6     7     19.7
## 3     8    14     15.1
```

The `my_cars` object itself occupies very little of R's memory.

```
my_cars %>%
  object.size() %>%
  print(units = "Kb")

## 3.2 Kb
```

C Benchmarking

Recall that in Section 2.2 we created a `tbl_dbi` called `trips` that is connected to a database table of Citi Bike trip rentals. In this example we illustrate how the ability of `dplyr` to offload certain computations to SQL can result in marked performance improvements, even on the same computer.

```
library(citibike)
bikes <- etl("citibike", dir = "~/dumps/citibike/",
             db = src_mysql_cnf("citibike"))
trips <- bikes %>%
  tbl("trips")
class(trips)
```

```
## [1] "tbl_dbi"  "tbl_sql"  "tbl_lazy" "tbl"
```

Previously, we used the following pipeline to compute the number of unique combinations of stations, days, and hours in the month of September 2013. In the code below, we make use of the lazy evaluation design of `dplyr` to push the computation to MySQL. Note that the functions in uppercase are MySQL functions—not R functions. The `collect()` verb is applied only after the database is queried so that R can count the number of resulting rows. Because MySQL is good at doing this type of operation, and only 167,258 rows of data are sent from MySQL to R, this computation takes only a few seconds.

```
system.time(
trips_sept <- trips %>%
  filter(YEAR(start_time) == 2013) %>%
  group_by(start_station_id, DAY(start_time), HOUR(start_time)) %>%
  summarize(N = n(),
            num_stations = COUNT(DISTINCT(start_station_id)),
            num_days = COUNT(DISTINCT(DAYOFYEAR(start_time)))) %>%
  collect()
)

##    user  system elapsed
##   0.365   0.010   1.844

nrow(trips_sept)

## [1] 167258
```

Conversely, we can use the `lubridate` package for assistance with dates, and the `collect()` function to bring the data into R for summarization. Note here that only the `filter()` operation is actually performed by MySQL, while the rest of the operations are performed in R.

```
library(lubridate)
system.time(
trips_sept <- trips %>%
  filter(YEAR(start_time) == 2013) %>%
  collect() %>%
  group_by(start_station_id, day(start_time), hour(start_time)) %>%
  summarize(N = n(),
            num_stations = n_distinct(start_station_id),
            num_days = n_distinct(yday(start_time)))
)

##    user  system elapsed
##  28.394   1.041  29.439

nrow(trips_sept)

## [1] 167258
```

This latter method is much slower since it has to transfer more than 1 million rows of data from MySQL to R, instead of only 167,258. The delay with the second method is noticeable enough to start a conversation with students about scalability.

D Using Amazon RDS

In this section we provide a brief tutorial explaining how to set up a medium database of taxi trip information on Amazon RDS (a cloud-based service) and populate it.

First, you must set up an Amazon Web Services account at `https://aws.amazon.com/rds/`. Our goal is to launch a new relational database service instance. In this example we will create a MySQL database that uses the Free Usage Tier (to avoid fees). In Figure 1, we show how to select the MySQL engine from among the available options.

Since we are simply testing this service, we select the "Dev/Test" usage case, which is the only one that is available under the Free Usage Tier (see Figure 2).

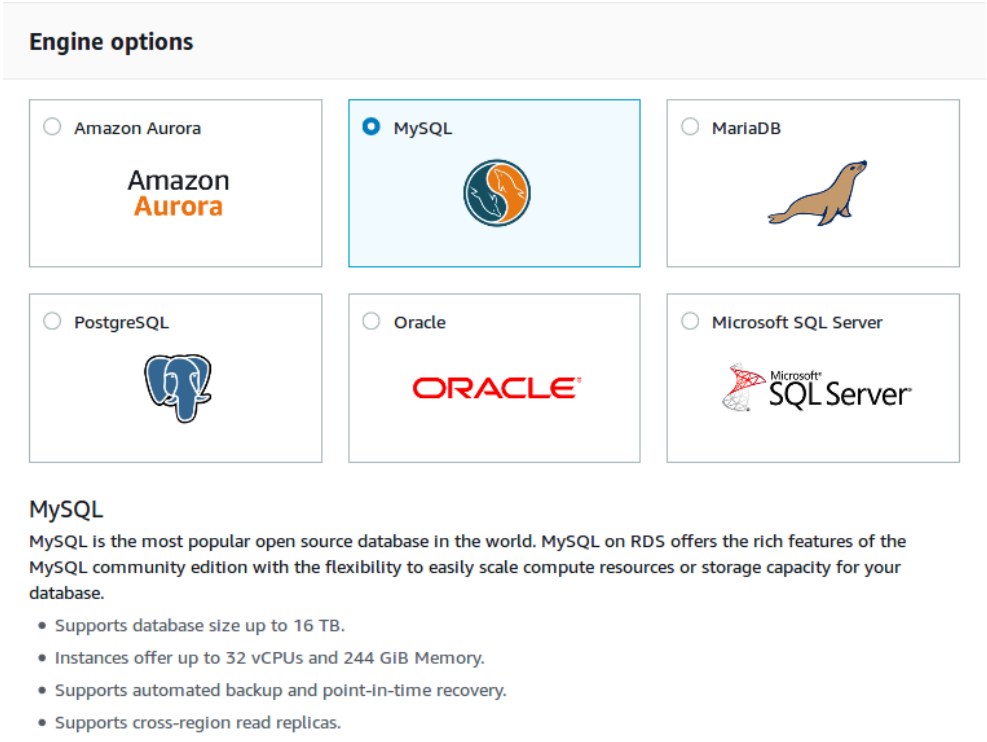

Figure 1: Amazon RDS

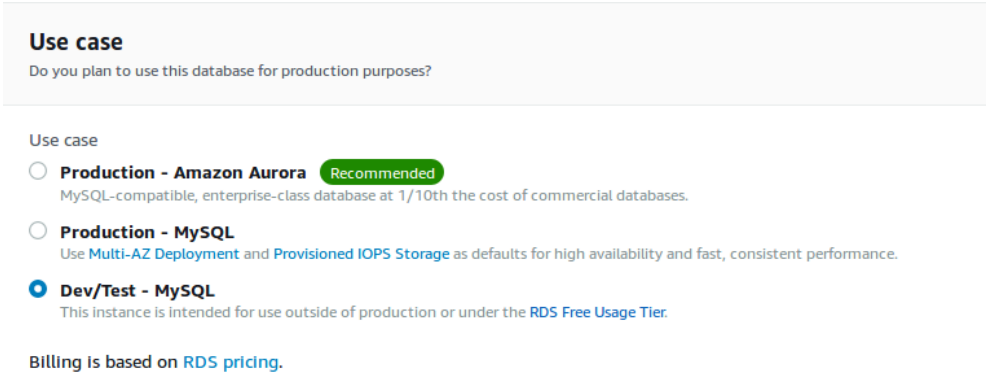

Figure 2: Amazon RDS

Next, in Figure 3 we allocate only minimal resources to this database instance. The `db.t2.micro` instance has only 1 CPU and 1 gigabyte of memory. This is the only allowable configuration in the Free Usage Tier.

In Figure 4, we elect to make our database publicly accessible. This is an important deviation from the default, which is to restrict access to a Virtual Private Cloud. Without selecting "Yes" here, we would not be able to connect to our database from our R client. Please consult the documentation on Amazon in order to fully understand your security settings. Note also that by default, public access is only granted from *your* IP address.

In the next step, we set up a username, password, and schema. These are specific to the MySQL instance on our cloud-based database server. After accepting all of the default options on the remaining screens, our instance will launch. This process creates a virtual MySQL server that is running on Amazon's servers. The hostname for that server is shown in your Instance dashboard under "Endpoint".

```
host <- "etl-test.cdc7tgkkqd0n.us-east-1.rds.amazonaws.com"
```

If we didn't set up a schema on the MySQL server called `nyctaxi` already, we can create one using the Terminal tab available in RStudio. Be sure to use the credentials for the MySQL instance that you specified.

```
mysql -h etl-test.cdc7tgkkqd0n.us-east-1.rds.amazonaws.com -u bbaumer -p -e
"CREATE DATABASE IF NOT EXISTS nyctaxi;"
```

Finally, we load the `nyctaxi` package and connect to our database instance.

```
library(nyctaxi)
db_rds <- src_mysql(dbname = "nyctaxi",
                    host = "etl-test.cdc7tgkkqd0n.us-east-1.rds.amazonaws.com",
                    user = "bbaumer",
                    password = "xxxxxxxx")
```

The `etl` grammar now allows us to easily populate the database.

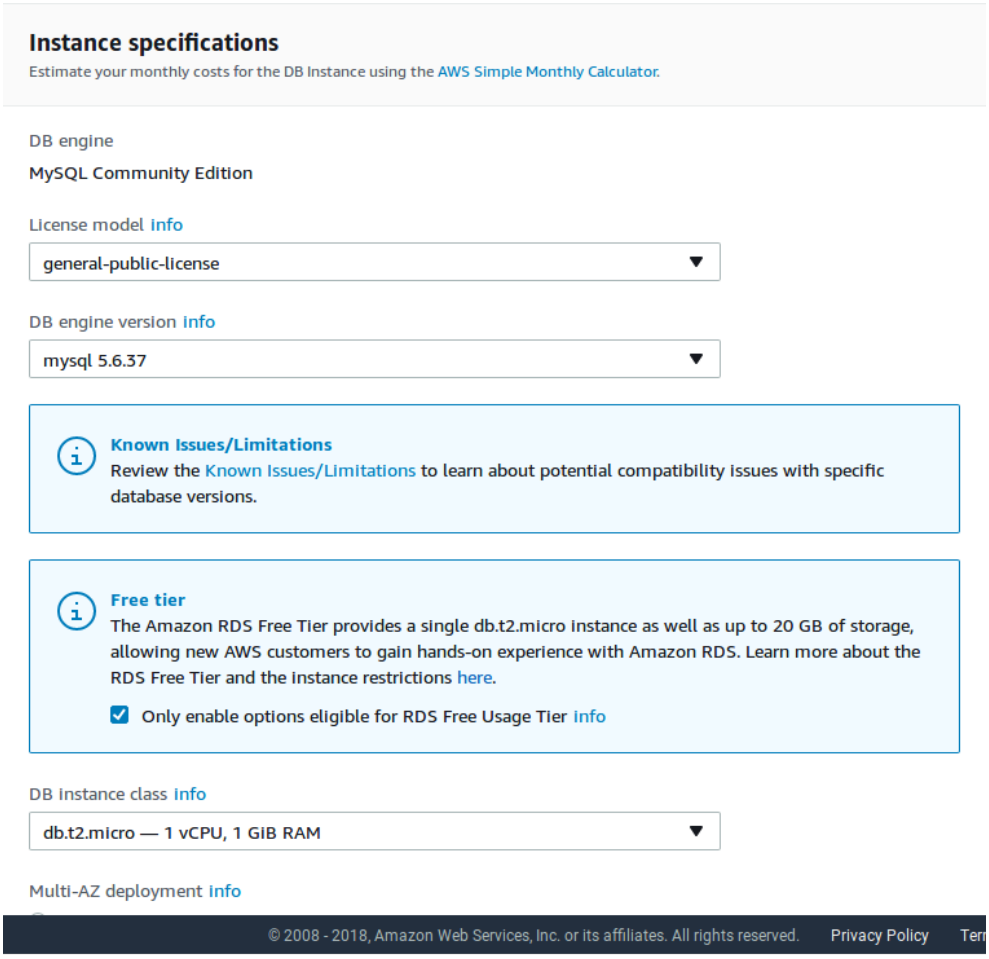

Figure 3: Amazon RDS

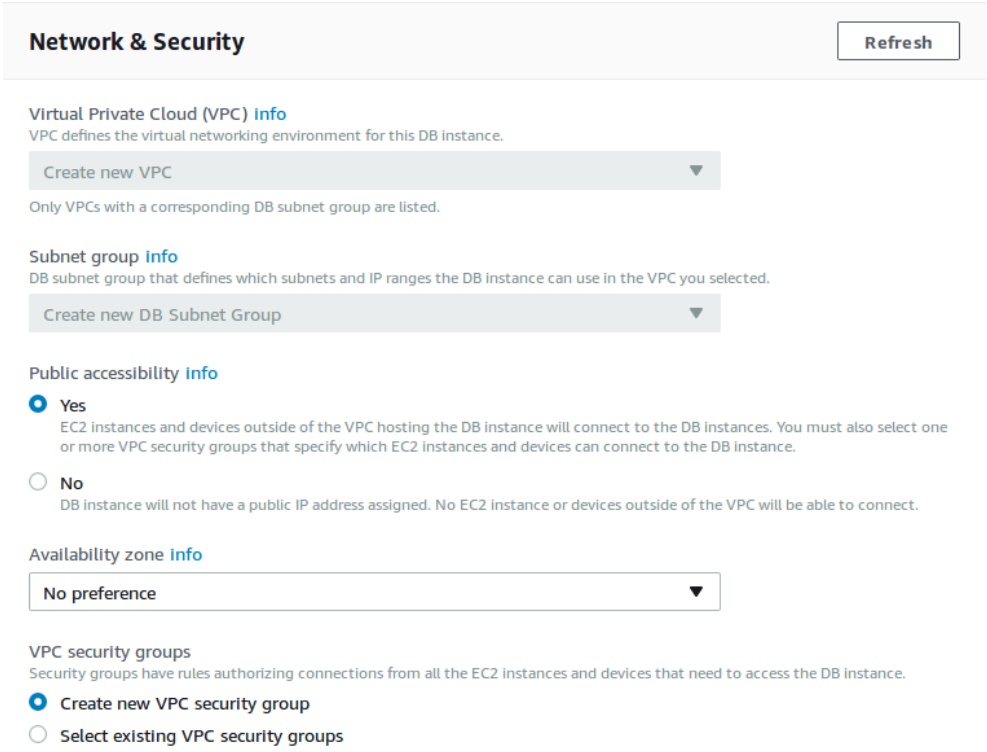

Figure 4: Amazon RDS

```
rides <- etl("nyctaxi", db = db_rds, dir = "~/dumps/nyctaxi")
rides %>%
  etl_update(years = 2014, months = 3)
```

References

Almquist, Z. W. (2010), 'US Census spatial and demographic data in R: The UScensus2000 suite of packages', *Journal of Statistical Software* **37**(6), 1–31.
URL: *http://www.jstatsoft.org/v37/i06/*

Anderson, G. B. & Eddelbuettel, D. (2017), 'Hosting Data Packages via drat: A Case Study with Hurricane Exposure Data', *The R Journal* **9**(1), 486–497.
URL: *https://journal.r-project.org/archive/2017/RJ-2017-026/index.html*

Ball, R. & Medeiros, N. (2012), 'Teaching integrity in empirical research: A protocol

for documenting data management and analysis', *The Journal of Economic Education* **43**(2), 182–189.
URL: *http://dx.doi.org/10.1080/00220485.2012.659647*

Baumer, B., Çetinkaya Rundel, M., Bray, A., Loi, L. & Horton, N. J. (2014), 'R Markdown: Integrating a reproducible analysis tool into introductory statistics', *Technology Innovations in Statistics Education* **8**(1).
URL: *http://escholarship.org/uc/item/90b2f5xh*

Baumer, B. S. (2017), *airlines: Historical On-time Flight Data*. R package version 0.2.2.9011.
URL: *http://github.com/beanumber/airlines*

Boettiger, C. (2015), 'An introduction to docker for reproducible research', *ACM SIGOPS Operating Systems Review* **49**(1), 71–79.

Boettiger, C., Chamberlain, S., Hart, E. & Ram, K. (2015), 'Building software, building community: lessons from the rOpenSci project', *Journal of Open Research Software* **3**(1).
URL: *https://openresearchsoftware.metajnl.com/articles/10.5334/jors.bu/*

Boettiger, C. & Eddelbuettel, D. (2017), 'An introduction to rocker: Docker containers for r', *arXiv preprint arXiv:1710.03675* .
URL: *https://arxiv.org/pdf/1710.03675*

Çetinkaya-Rundel, M. & Rundel, C. (2017), 'Infrastructure and tools for teaching computing throughout the statistical curriculum', *The American Statistician* (just accepted).
URL: *https://peerj.com/preprints/3181.pdf*

Claerbout, J. (1994), Hypertext documents about reproducible research, Technical report, Stanford University.
URL: *http://sepwww.stanford.edu/sep/jon/nrc.html*

Donoho, D. L. (2010), 'An invitation to reproducible computational research', *Biostatistics* **11**(3), 385–388.
URL: *https://academic.oup.com/biostatistics/article/11/3/385/257703*

Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M. & Stodden, V. (2009), 'Reproducible research in computational harmonic analysis', *Computing in Science & Engineering* **11**(1).

Eckel, S. P. & Peng, R. D. (2009), 'Interacting with local and remote data repositories using the stashr package', *Computational Statistics* **24**(2), 247–254.
URL: *https://link.springer.com/content/pdf/10.1007/s00180-008-0124-x.pdf*

Eddelbuettel, D. & François, R. (2011), 'Rcpp: Seamless R and C++ integration', *Journal of Statistical Software* **40**(8), 1–18.
URL: *http://www.jstatsoft.org/v40/i08/*

Eglen, S. J., Marwick, B., Halchenko, Y. O., Hanke, M., Sufi, S., Gleeson, P., Silver, R. A., Davison, A. P., Lanyon, L., Abrams, M. et al. (2017), 'Toward standard practices for sharing computer code and programs in neuroscience', *Nature Neuroscience* **20**(6), 770–773.
URL: *https://www.biorxiv.org/content/early/2017/02/28/045104*

Fuentes, M. (2016), 'Reproducible research in JASA', *AMSTAT News* .
URL: *http://magazine.amstat.org/blog/2016/07/01/jasa-reproducible16/*

Hardin, J., Hoerl, R., Horton, N. J., Nolan, D., Baumer, B., Hall-Holt, O., Murrell, P., Peng, R., Roback, P., Temple Lang, D. et al. (2015), 'Data science in statistics curricula: Preparing students to 'think with data'', *The American Statistician* **69**(4), 343–353.
URL: *http://www.tandfonline.com/doi/abs/10.1080/00031305.2015.1077729*

Horton, N. J., Baumer, B. S. & Wickham, H. (2015), 'Setting the stage for data science: integration of data management skills in introductory and second courses in statistics', *Chance* **28**(2).
URL: *http://chance.amstat.org/2015/04/setting-the-stage/*

Ioannidis, J. P. (2005), 'Why most published research findings are false', *PLoS medicine* **2**(8), e124.
URL: *http://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.0020124*

Knuth, D. E. (1984), 'Literate programming', *The Computer Journal* **27**(2), 97–111.

Marwick, B. (2017), 'Computational reproducibility in archaeological research: Basic principles and a case study of their implementation', *Journal of Archaeological Method and Theory* **24**(2), 424–450.
URL: *https://osf.io/preprints/socarxiv/q4v73/download?format=pdf*

Peng, R. D. & Dominici, F. (2008), *Statistical methods for environmental epidemiology with R*, Springer: New York.
URL: *https://link.springer.com/content/pdf/10.1007/978-0-387-78167-9.pdf*

Sievert, C. (2014), 'Taming PITCHf/x data with pitchRx and XML2R', *The R Journal* **6**(1).
URL: *http://journal.r-project.org/archive/2014-1/sievert.pdf*

Walker, K. & Rudis, B. (2017), *Tigris: Load Census TIGER/Line Shapefiles into R.* R package version 0.3.3.
URL: *https://CRAN.Rproject.org/package=tigris*

Wickham, H. (2013), *hflights: Flights that departed Houston in 2011.* R package version 0.1.
URL: *https://CRAN.R-project.org/package=hflights*

Wickham, H. (2015), *R packages*, O'Reilly Media, Inc.: Sebastopol, CA.
URL: *http://r-pkgs.had.co.nz/*

Wickham, H. (2016), *nycflights13: Flights that Departed NYC in 2013.* R package version 0.2.0.
URL: *https://CRAN.R-project.org/package=nycflights13*

Williams, V. V. (2012), Multiplying matrices faster than Coppersmith-Winograd, *in* H. J. Karloff & T. Pitassi, eds, 'Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012', ACM, pp. 887–898.
URL: *http://doi.acm.org/10.1145/2213977.2214056*